# Cyclone Automated Control SDK

## Developer's Manual


Visit us on the web at http://www.pemicro.com

# Document Version History

| Version | Date | Notes |
|---|---|---|
| 1.3 | 1 July 1, 2008 | Added documentation for the included Cyclone Launch application |
| 1.2 | 14 April 2008 | Added documentation for C# support |
| 1.1 | 13 March 2008 | Added documentation for software licensing<br>Added documentation for “close_all_ports” function |
| 1.0 | 10 January 2008 | Initial document |

# CONTENTS

# 1 Introduction

Thank you for installing P&E's Cyclone Automated Control SDK!

This development kit allows you to create an application on the PC that can directly control one or more P&E Cyclone units. These interface routines are designed to be compiled into visual and non visual applications running on Windows 95, 98, ME, NT, 2000, or XP.

With the routines included in this SDK, you can use a PC to start Cyclone programming operations, maintain multiple standalone images, and automate your production programming process like never before!

The actual interface routines are located in the "CYCLONE_CONTROL.DLL" 32 bit DLL file. The DLL is callable from almost any 32-bit Windows development environment. Since the way the DLL is called varies depending on the compiler used, you are provided with the DLL interface code and sample applications for each of the following compilers:

| | |
|---|---|
| Borland Delphi 2.0+ (Pascal) | - Visual Application |
| Microsoft Visual C++ 5.0+ | - Visual MFC Application |
| Microsoft Visual C# 2005+ | - Visual Application |

The sample applications come with project and workspaces defined for ease of use. Just open the project/workspace in your compiler and you should be able to build the sample application without any modifications. The sample applications come pre compiled with ICONS, so you can run them before jumping into the code. The callable interface routines are defined in:

INSTALLDIR\Delphi20\multiple_cyclone_programming.pas
INSTALLDIR\msvc50\multiple_cyclone_programming.h
INSTALLDIR\msvcsharp2005\visual_sap_control\
multiple_cyclone_programming.cs

If you add the above mentioned interface file (and associated .CPP file for the C++ compilers) to your own application, you will be able to directly access all of the routines in the SDK. Remember that the DLL file must be in the executable's directory or system directory for the application to control the Cyclone.

## 1.1 Library Files Included

Due to the various calling conventions that currently exist, P&E provides the developer with two different versions of the DLL. The example programs include header files which use the “cdecl” calling convention. The user must modify these header files if they wish to use the “stdcall” version of the library.

| **Library file** | **Calling Convention** |
|---|---|
| cyclone_control.dll | cdecl |
| cyclone_control_cdecl.dll[1] | cdecl |
| cyclone_control_stdcall.dll | stdcall |

[1] Note that “cyclone_control_cdecl.dll” is identical to “cyclone_control.dll”

## 1.2 P&E Compatible Hardware

The following lists the P&E hardware compatible with the Cyclone Automated Control SDK. To ensure proper operations, P&E recommends upgrading all Cyclone units to the latest firmware prior to using this SDK.

- Cyclone PRO
- Cyclone MAX

## 1.3 SDK Versions

The Cyclone Automated Control SDK product is split into three separate versions. While this manual is written to support all three versions, it is important to note the following limitations:

Basic Edition

- DLL only supports controlling a single Cyclone unit
- DLL will only execute the first image stored on a Cyclone (using the START_execute_all_commands function call)
- DLL does not support the “START_dynamic_program_bytes” function
- DLL does not support the “compare_image_with_file” function
- DLL does not support the “erase_all_cyclone_images” function
- DLL does not support the “add_image_to_cyclone” function
- DLL does not support the “update_image_with_file” function
- RS232 / Ethernet communication protocols are not included

Professional Edition

- DLL supports controlling of up to 3 Cyclone units simultaneously
- DLL supports all available function calls
- RS232/ Ethernet communication protocols are not included

Enterprise Edition

- DLL supports controlling of an unlimited number of Cyclone units simultaneously
- DLL supports all available function calls
- RS232/ Ethernet communication protocols are included

## 2 Software License

This software and accompanying documentation are protected by United States Copyright law and also by International Treaty provisions. Any use of this software in violation of copyright law or the terms of this agreement will be prosecuted.

This software is copyrighted by P&E Microcomputer Systems, Inc. Copyright notices have been included in the software.

P&E Microcomputer Systems authorizes you to make archival copies of this software for the sole purpose of back-up and protecting your investment from loss. Under no circumstances may you copy this software or documentation for the purpose of distribution to others. Under no conditions may you remove the copyright notices from this software or documentation.

This software may be used by one person on as many computers as that person uses, provided that the software is never used on two computers at the same time. P&E expects that group programming projects making use of this software will purchase a copy of the software and documentation for each user in the group. Contact P&E for volume discounts and site licensing agreements.

With respect to the physical media provided within, P&E Microcomputer Systems warrants the same to be free of defects in materials and workmanship for a period of 30 days from the date of receipt. If you notify us within the warranty period, P&E Microcomputer Systems will update the defective media at no cost.

P&E Microcomputer Systems does not assume any liability for the use of this software beyond the original purchase price of the software. In no event will P&E Microcomputer Systems be liable for additional damages, including any lost profits, lost savings or other incidental or consequential damages arising out of the use or inability to use these programs, even if P&E Microcomputer Systems has been advised of the possibility of such damage.

## 2.1 P&E Software Licensing System

When the DLL is used for the first time, a popup form will be shown prompting the user to enter the relevant information to activate the software. Please refer to the original packaging for the necessary installation code. Please follow the on-screen instructions to activate the product.

In a situation where you would like to change versions, (e.g. upgrading from Basic edition to Professional edition) please perform the following procedure:

1. Uninstall the current version of the software
2. Search your PC for the “pemicro.lic” file*. Backup or delete this file.
3. Install the new version of the software
4. Proceed with the software activation process using the new installation code that you should have received

* The “pemicro.lic” file is found in the same directory where the DLL is located. Normally this is found in the Windows system directory:

- \WINDOWS\SYSTEM for Windows 95, Windows 98, Windows ME
- \WINNT\SYSTEM32 for Windows 2000, Windows NT
- \WINDOWS\SYSTEM32 for Windows XP, Windows Vista

# 3 Getting Started

This section outlines the steps you need to take to begin developing your own custom application and offers tips and suggestions to get the Cyclone Automated Control SDK working with your P&E hardware smoothly.

## 3.1 Example Programs

Located in the installation directory of the SDK, you will find two example programs that you can use as a reference for your own application. The examples are located in the following directories:

| | |
|---|---|
| INSTALLDIR\Delphi20 | - Delphi 2.0+ Example Visual Application |
| INSTALLDIR\msvc50 | - MSVC 5.0+ Example MFC Visual Application |
| INSTALLDIR\msvcsharp2005 | - MSVC# 2005+ Example Visual Application |

These example programs are a valuable reference to use when starting your own custom application.

## 3.2 Starting your own project

To gain access to the functions available in the SDK, the following files need to be added to the new project workspace:

Delphi 2.0+ Projects

INSTALLDIR\Delphi20\multiple_cyclone_programming.pas

All other source files which will call functions from the SDK should include the above file using the Delphi “uses” command.

MSVC 5.0+ Projects

INSTALLDIR\msvc50\multiple_cyclone_programming.h
INSTALLDIR\msvc50\multiple_cyclone_programming.cpp

All other source files which will call functions from the SDK should include the above header file with the C/C++ #include directive.

MSVC# 2005+ Projects

INSTALLDIR\msvcsharp2005\multiple_cyclone_programming.cs

Once added to the project workspace, the namespace of the file may optionally be modified to match the namespace of your project.

## 3.3 Initialization

Loading the DLL (C/C++ Projects only)

Before calling any routines from the SDK, the DLL must be loaded into memory. To do this, the following function has been provided in the included header files. Refer to Chapter 4 of this manual for a detailed description of this function.

*load_dll( )*

For Delphi (Pascal) and C# users, this process is transparent for the user and no action is required.

Enumerate all ports

After loading the DLL, a call to the following function is required in order to properly initialize all devices. This function should only be called once, typically at the beginning of the application.

*enumerate_all_ports( )*

Connect to the P&E hardware interface

The next step is to establish communications with the P&E Cyclone unit. This is accomplished with the following function call:

*connect_to_cyclonepromax( )*

Refer to Chapter 4 of this manual for a detailed description of this function. This call returns the handle to the Cyclone unit which is used in all other routines in the SDK to identify the Cyclone. Note that the special case of a return value of 0 indicates an error contacting the Cyclone. This function will be called once for each Cyclone.

## 3.4 Finalization

Before closing the application, it is recommended that the session with the P&E hardware be terminated and the DLL unloaded from memory. Note that the unloading of the DLL should be the last call made to the SDK.

These calls should always be made before the application closes:

*disconnect_from_cyclonepromax( );*
*close_all_ports( );*
*unload_dll( );*

Note that the “unload_dll” call is only required for C/C++ applications.

## 3.5 Initial Cyclone Setup

The Cyclone Image Creation Utility software, which is included with each Cyclone unit, is used to create the standalone images that will be stored in the non-volatile memory of the Cyclone. These images contain the FLASH / EEPROM programming algorithms, the actual binary data to be programmed, the sequence of programming operations, and user specified Cyclone settings.

Prior to using the Cyclone Automated Control SDK, these standalone images need to be created. Please refer to the user’s manual of your Cyclone unit for more information on standalone images and image creation.

## 3.6 Typical Usage

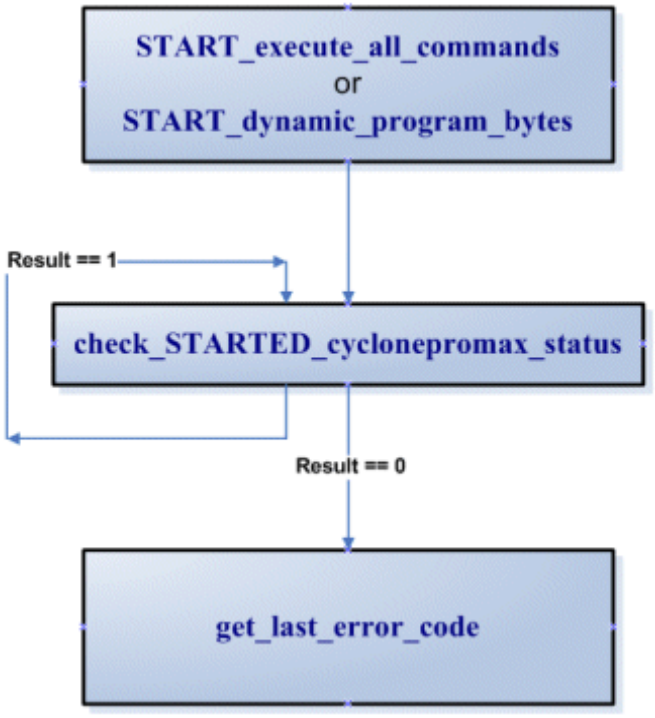


**Figure 3-1: Typical programming procedure flow chart**

Figure 3-1 describes the most common sequence of calls to the DLL after successfully connecting to the Cyclone unit.

**Step 1**: Initiate programming operations. “START_execute_all_commands” carries out the programming operations defined in the stand-alone image stored on the Cyclone unit. “START_dynamic_program_bytes” allows the user to manually specify the data as well as the memory address of the programming.

**Step 2** : Wait for programming completion. Note that no error checking is provided by the “check_STARTED_cyclonepromax_status” call. A result of 0 will be returned even if an error has occurred.

**Step 3** : Retrieve the error code from the Cyclone unit to determine if the programming was successful.

# 4 Application Programming Interface (API)

This chapter describes the API of the “CYCLONE_CONTROL.DLL” in detail. Note that certain functions in the API are not supported in the Basic Edition of the DLL. Please refer to section 1.3 of this manual for more details.

In each section, the function prototypes are given in the following order:
Delphi
C/C++
C#

## 4.1 DLL Loading / Unloading Calls

### 4.1.1 load_dll

***bool load_dll(void);***

Loads the CYCLONE_CONTROL.DLL into memory and gives the user access to all of the functions available in the library. **This routine must be called before any of the other routines can be called.** This routine is only required for C/C++ applications and is defined in the file “multiple_cyclone_programming.cpp”.

| | |
|---|---|
| @returnvalue | True if the load was successful<br>False otherwise |

### 4.1.2 unload_dll

***void unload_dll (void);***

Unloads the DLL loaded with load_dll( ). This call should be made before the application starts to unload itself. This routine is only required for C/C++ applications and is defined in the file “multiple_cyclone_programming.cpp”

### 4.1.3 enumerate_all_ports

***procedure enumerate_all_ports;***
***void enumerate_all_ports(void);***
***void enumerate_all_ports(void);***

Performs all necessary initialization in order to successfully communicate with a Cyclone unit. This function needs to be called once, before the first call to “connect_to_cyclonepromax”.

### 4.1.4 close_all_ports

***procedure close_all_ports;***
***void close_all_ports(void);***
***void close_all_ports(void);***

This call closes all open Cyclone units (if any) and frees all dynamic memory used by the DLL. This function should be called before the user application is closed.

### 4.1.5 version

***function version : pchar;***
***char * version(void);***
***String version(void);***

| | |
|---|---|
| @returnvalue | Returns a pointer to a null-terminated string containing the version number of the DLL. |

## 4.2 Cyclone Connecting / Disconnecting Calls

### 4.2.1 connect_to_cyclonepromax

***function connect_to_cyclonepromax(port_type : longword; identifier_type : longword; port_identifier : pchar) : longword;***
***unsigned long connect_to_cyclonepromax(unsigned long port_type, unsigned long identifier_type, char* port_identifier);***
***UInt32 connect_to_cyclonepromax(UInt32 port_type, UInt32 identifier_type, String port_identifier);***

Opens a session with a Cyclone unit. The handle is returned by this function is passed as a parameter to all other functions in the SDK. Note that the DLL does not support multiple Cyclones connected via the serial port. If you require more than one Cyclone to use a serial port connection, P&E recommends using the RS232 communication protocols available in the Enterprise Edition of the SDK.

| | |
|---|---|
| @param port_type | Specifies how the Cyclone is currently connected to the host PC. One of three possible values:<br><br>*PortType_USB*<br>*PortType_Ethernet*<br>*PortType_Serial*<br><br>These constants are defined in the header files of the example projects. |
| @param identifier_type | Specifies whether to identify the Cyclone by its IP address or its name. One of two possible values:<br><br>*Open_by_IP_Address*<br>*Open_by_Name*<br><br>These constants are defined in the header files of the example projects. |
| @param port_identifier | A pointer to a null-terminated character string which identifies the Cyclone. If identifying by IP address, the string should be in the format of xxx.xxx.xxx.xxx, where xxx = 0…255. If identifying by name, the string should contain the name of the Cyclone unit. |
| @returnvalue | The handle to the opened Cyclone unit. A |

| | |
|---|---|
| | return value of 0 indicates a failure to connect to the specified Cyclone unit. |

### 4.2.2 connect_to_cyclonepromax_by_ip

***function connect_to_cyclonepromax_by_ip(port_identifier : pchar) : longword;***
***unsigned long connect_to_cyclonepromax_by_ip(char * port_identifier);***
***UInt32 connect_to_cyclonepromax_by_ip(String port_identifier);***

This function is a legacy call and should not be used for new applications. “connect_to_cyclonepromax” should be used instead. This call opens a session with a Cyclone unit that is connected via Ethernet by its IP address. The handle that is returned by this function is passed as a parameter to all other calls in the SDK.

| | |
|---|---|
| @param port_identifier | A pointer to a null-terminated character string in the format xxx.xxx.xxx.xxx, where xxx = 0…255. |
| @returnvalue | The handle to the opened Cyclone unit. A return value of 0 indicates a failure to connect to the Cyclone unit at that IP address. |

### 4.2.3 disconnect_from_cyclonepromax

***function disconnect_from_cyclonepromax(cyclonepromaxhandle : longword) : boolean;***
***bool disconnect_from_cyclonepromax(unsigned long cyclonepromaxhandle);***
***bool disconnect_from_cyclonepromax(UInt32 cyclonepromaxhandle);***

When processing is complete, the session with the Cyclone unit should be terminated. Calling this routine with the appropriate handle terminates the session. All Cyclone units should be disconnected when the application is closed.

| | |
|---|---|
| @param cyclonepromaxhandle | The handle of the Cyclone unit to have its session terminated |
| @returnvalue | True if the session terminated successfully<br>False otherwise<br><br>Examples of situations which could return a value of false:<br><br>Invalid handle number<br>Cyclone unit specified is not currently open |

### 4.2.4 set_local_machine_ip_number

***procedure set_local_machine_ip_number(ip_number : pchar);***
***void set_local_machine_ip_number(char* ip_number);***
***void set_local_machine_ip_number(String ip_number);***

If a machine has more than one network card, this routine may be called to indicate the IP address of the network card which should be used during communications. This should be called prior to calling any other routines.

| | |
|---|---|
| @param ip_number | A pointer to a null-terminated character string in the format xxx.xxx.xxx.xxx, where xxx = 0…255. |

## 4.3 Controlling Cyclone Programming

### 4.3.1 START_execute_all_commands

***function START_execute_all_commands(cyclonepromaxhandle : longword; image_id : byte) : boolean;***
***bool START_execute_all_commands(unsigned long cyclonepromaxhandle, unsigned char image_id);***
***bool START_execute_all_commands(UInt32 cyclonepromaxhandle, byte image_id);***

A Cyclone unit may have several independent programming images in non-volatile memory. A programming image contains the programming algorithms, binary data, and programming sequence. Calling this routine instructs the Cyclone unit to start execution of a particular image. After invoking this call, the “check_STARTED_cyclonepromax_status” function should be used to wait for completion.

| | |
|---|---|
| @param cyclonepromaxhandle | The handle of the Cyclone unit to begin programming operations |
| @param image_id | Used to select which image stored on the Cyclone unit to use. If a Cyclone only stores one image, this parameter should be set to 1.<br><br>The valid range of this parameter is from 1 to the number of images in the Cyclone unit. |
| @returnvalue | True if the programming process has started successfully.<br><br>False otherwise. |

### 4.3.2 START_dynamic_program_bytes

***function START_dynamic_program_bytes(cyclonepromaxhandle : longword; target_address : longword; data_length : word; buffer : pointer) : boolean;***
***bool START_dynamic_program_bytes( unsigned long cyclonepromaxhandle, unsigned long target_address, unsigned short data_length, char *buffer);***
***bool START_dynamic_program_bytes(UInt32 cyclonepromaxhandle, UInt32 target_address, UInt16 data_length, byte[ ] buffer);***

Sometimes, in addition to the large amount of static data being programmed into a target from the Cyclone, it is advantageous for the calling application to program small sections of unique data dynamically. Examples of this include date/time, serial number, MAC addresses, and lot numbers. This routine allows the user to program such dynamic data into the processor.

This function is valid to be called only after a programming image has been programmed into the target (once START_execute_all_commands has completed). Call the “check_STARTED_cyclonepromax_status” function to wait for completion.

| | |
|---|---|
| @param cyclonepromaxhandle | The handle of the Cyclone unit to begin dynamic programming. |
| @param target_address | The first memory address of the target processor where the dynamic data should be written. |
| @param data_length | The total number of bytes to be written. This parameter cannot be greater than 255. |
| @param buffer | Pointer to the array which holds the data to be written. |
| @returnvalue | True if the programming process has started successfully.<br><br>False otherwise. |

### 4.3.3 check_STARTED_cyclonepromax_status

***function check_STARTED_cyclonepromax_status(***
***cyclonepromaxhandle : longword) : longword;***
***unsigned long check_STARTED_cyclonepromax_status(unsigned long cyclonepromaxhandle);***
***UInt32 check_STARTED_cyclonepromax_status(UInt32 cyclonepromaxhandle);***

Checks to see if the Cyclone unit has completed a programming operation started with either the “START_execute_all_commands” or “START_dynamic_program_bytes” routines.

After this call returns with completed value, “get_last_error_code” should be called to determine the programming result.

| | |
|---|---|
| @param cyclonepromaxhandle | The handle of the Cyclone unit to perform a status check on. |
| @returnvalue | 1 = Currently programming<br>0 = Completed (with or without error) |

### 4.3.4 dynamic_read_bytes

***function dynamic_read_bytes(cyclonepromaxhandle : longword; target_address : longword; data_length: word; buffer: pointer) : boolean;***
***bool dynamic_read_bytes(unsigned long cyclonepromaxhandle, unsigned long target_address, unsigned short data_length, char *buffer);***
***bool dynamic_read_bytes(UInt32 cyclonepromaxhandle, UInt32 target_address, UInt16 data_length, byte [ ] buffer);***

Reads a specified number of bytes from a specified memory address of the target processor. This call is only valid after performing a "START_execute_all_commands" function.

| | |
|---|---|
| @param cyclonepromaxhandle | The handle of the Cyclone unit that will perform the dynamic read. |
| @param target_address | The first memory address of the target processor where the data will be read. |
| @param data_length | The number of total bytes to read from the target processor |
| @param buffer | A pointer to the array where the data read will be stored. This array must have been allocated prior to calling this routine. |
| @returnvalue | True if the data was successfully read<br>False otherwise |

### 4.3.5 get_last_error_code

***function get_last_error_code(cyclonepromaxhandle : longword) : word;***
***unsigned short get_last_error_code(unsigned long cyclonepromaxhandle);***
***UInt16 get_last_error_code(UInt32 cyclonepromaxhandle);***

Returns the last error code recorded by a Cyclone unit. Refer to section 5 for all possible error codes.

| | |
|---|---|
| @param cyclonepromaxhandle | The handle of the Cyclone unit from which to request the error code. |
| @returnvalue | The last error code of the specified Cyclone unit. A return value of 0 indicates that no error occurred during the programming process. |

### 4.3.6 get_last_error_addr

***function get_last_error_addr(cyclonepromaxhandle : longword) : longword;***
***unsigned long get_last_error_addr(unsigned long cyclonepromaxhandle);***
***UInt32 get_last_error_address(UInt32 cyclonepromaxhandle);***

If the “get_last_error_code” function returns a non-zero value (indicating an error has occurred), this routine can be used to query the address where the error occurred.

| | |
|---|---|
| @param cyclonepromaxhandle | The handle of the Cyclone unit from which to request the error address. |
| @returnvalue | The memory address where the last programming error occurred. |

### 4.3.7 pro_set_active_security_code

***function pro_set_active_security_code(cyclonepromaxhandle: longword; buffer : pointer) : boolean;***
***bool pro_set_active_security_code(unsigned long cyclonepromaxhandle, char *buffer);***
***bool pro_set_active_security_code(UInt32 cyclonepromaxhandle, byte [ ] buffer);***

Sets the security code used by the Cyclone Pro to enter monitor mode for HC908 targets. Once a blank HC908 device is programmed, this security code must be known in order to enter monitor mode without erasing the device.

This function can be useful if testing needs to be performed after programming (requiring the HC908 device to be powered off and powered back on), and later you need to program a serial number, for example.

| | |
|---|---|
| @param cyclonepromaxhandle | The handle of the Cyclone unit that will have its HC908 security code set. |
| @param buffer | A pointer to a character array which contains the security code. Note that the character array should not include the dashes between bytes.<br><br>For example, a security code of “01-02-03-04-05-06-07-08” should be represented as “0102030405060708” in the array. |
| @returnvalue | True if no error occurred<br>False otherwise |

## 4.4 Configuration / Image Maintenance Calls

### 4.4.1 reset_cyclonepromax

***function reset_cyclonepromax(cyclonepromaxhandle : longword; reset_delay_in_ms : longword) : boolean;***
***bool reset_cyclonepromax(unsigned long cyclonepromaxhandle, unsigned long reset_delay_in_ms);***
***bool reset_cyclonepromax(UInt32 cyclonepromaxhandle, UInt32 reset_delay_in_ms);***

Used to reset the Cyclone hardware. This routine is a legacy call and does not need to be called by the application.

| | |
|---|---|
| @param cyclonepromaxhandle | The handle of the Cyclone unit that will be reset. |
| @param reset_delay_in_ms | The reset delay, specified in milliseconds. The delay should be at least 2500 ms. |
| @returnvalue | True if reset was successful<br>False otherwise |

### 4.4.2 get_firmware_version

***function get_firmware_version(cyclonepromaxhandle : longword) : pchar;***
***char *get_firmware_version(unsigned long cyclonepromaxhandle);***
***String get_firmware_version(UInt32 cyclonepromaxhandle);***

Used to determine the current firmware version of the Cyclone unit.

| | |
|---|---|
| @param cyclonepromaxhandle | The handle of the Cyclone unit of which to read the firmware version. |
| @returnvalue | Returns a pointer to a null-terminated character string containing the firmware version. |

### 4.4.3 get_image_description

***function get_image_description(cyclonepromaxhandle : longword; image_id : byte) : pchar;***
***char *get_image_description(unsigned long cyclonepromaxhandle, unsigned char image_id);***
***String get_image_description(UInt32 cyclonepromaxhandle, byte image_id);***

Checks the description of a particular image stored on the Cyclone unit. This description is specified by the user when the image is first created.

| | |
|---|---|
| @param cyclonepromaxhandle | The handle of the Cyclone unit to get an image description from |
| @param image_id | Used to select which image stored on the Cyclone unit to read the description from. If a Cyclone only stores one image, this parameter should be set to 1.<br><br>The valid range of this parameter is from 1 to the number of images in the Cyclone unit. |
| @returnvalue | A pointer to a null-terminated string which contains the image description |

### 4.4.4 compare_image_with_file

***function compare_image_with_file(cyclonepromaxhandle : longword; aFile : pchar; image_id : byte) : boolean;***
***bool compare_image_with_file(unsigned long cyclonepromaxhandle, char *aFile, unsigned char image_id);***
***bool compare_image_with_file(UInt32 cyclonepromaxhandle, String aFile, byte image_id);***

Compares an image stored on a Cyclone unit against a .SAP file created with the Cyclone Image Creation Utility.

| | |
|---|---|
| @param cyclonepromaxhandle | The handle of the Cyclone unit that will have its image compared |
| @param aFile | A pointer to a null-terminated character string which contains the full path to the .SAP file that will be compared |
| @param image_id | Used to select which image stored on the Cyclone unit to compare against. If a Cyclone only stores one image, this parameter should be set to 1.<br><br>The valid range of this parameter is from 1 to the number of images in the Cyclone unit. |
| @returnvalue | True if the image and the .SAP file match<br>False otherwise<br><br>Note that a false will also be returned if an error occurred during communications between the PC and the Cyclone unit. |

### 4.4.5 erase_all_cyclone_images

***function erase_all_cyclone_images(cyclonepromaxhandle : longword) : boolean;***
***bool erase_all_cyclone_images(unsigned long cyclonepromaxhandle);***
***bool erase_all_cyclone_images(UInt32 cyclonepromaxhandle);***

Erases all images stored on the Cyclone unit. This function should not be constantly called, as this will shorten the lifespan of the Cyclone unit's internal memory. It is recommended that the user make use of the "compare_image_with_file" function first to determine if an erase is indeed necessary. (e.g. if the images on the Cyclone unit do not match the latest images on a server)

| | |
|---|---|
| @param cyclonepromaxhandle | The handle of the Cyclone unit that will have its images erased |
| @returnvalue | True if the erasure was successful<br>False otherwise |

### 4.4.6 add_image_to_cyclone

***function add_image_to_cyclone(cyclonepromaxhandle : longword; aFile : pchar) : longword;***
***unsigned long add_image_to_cyclone(unsigned long cyclonepromaxhandle, char *aFile);***
***UInt32 add_image_to_cyclone(UInt32 cyclonepromaxhandle, String aFile);***

Adds a specified stand-alone programming image into the Cyclone's onboard memory. The image files have a .SAP file extension and are created with the Cyclone Image Creation Utility. Since the Cyclone has limited onboard memory, the user may wish to erase all existing images first, using the "erase_all_cyclone_images" function.

| | |
|---|---|
| @param cyclonepromaxhandle | The handle of the Cyclone unit that will accept the new image |
| @param aFile | A pointer to a null-terminated character string which contains the full path to the .SAP file to be added. |
| @returnvalue | The image number of the image that was just added. This number is used as the "image_id" parameter for some function calls.<br><br>A return value of "0" indicates an error has occurred during the process |

### 4.4.7 update_image_with_file

***function update_image_with_file(cyclonepromaxhandle : longword; aFile : pchar) : boolean;***
***bool update_image_with_file(unsigned long cyclonepromaxhandle, char *aFile);***
***bool update_image_with_file(UInt32 cyclonepromaxhandle, String aFile);***

Updates an existing image stored on the Cyclone unit with a new .SAP image file on the host PC. This function operates properly only if there exists a single image on the Cyclone.

This call is equivalent to calling “erase_all_cyclone_images” followed by “add_image_to_cyclone”.

| | |
|---|---|
| @param cyclonepromaxhandle | The handle of the Cyclone unit that will have its image updated |
| @param aFile | A pointer to a null-terminated character string which contains the full path to the .SAP file to be added. |
| @returnvalue | True if the image updated is successful<br>False otherwise |

### 4.4.8 count_cyclonepromax_images

***function count_cyclonepromax_images(cyclonepromaxhandle : longword) : byte;***
***unsigned char count_cyclonepromax_images(unsigned long cyclonepromaxhandle);***
***byte count_cyclonepromax_images(UInt32 cyclonepromaxhandle);***

Returns the number of stand-alone programming images currently stored in the Cyclone’s onboard memory.

| | |
|---|---|
| @param cyclonepromaxhandle | The handle of the Cyclone unit to query for the image count |
| @returnvalue | The total number of images stored on the Cyclone |

### 4.4.9 toggle_power_no_background_entrance

***function toggle_power_no_background_entrance(***
***cyclonepromaxhandle : longword) : boolean;***
***bool toggle_power_no_background_entrance(unsigned long***
***cyclonepromaxhandle);***
***bool toggle_power_no_background_entrance(UInt32***
***cyclonepromaxhandle);***

Toggles the output target power of the Cyclone unit. The signals used to enter background mode are also tristated during this time. This call is normally used to power down and power up the target so that the processor is running normally after programming.

The Cyclone's jumper settings must be configured properly in order for the Cyclone's output target power to reach the processor. Please refer to the Cyclone manual for more details.

*Note: This function has no effect for the Cyclone Max, as the output power circuitry is not implemented.*

| | |
|---|---|
| @param cyclonepromaxhandle | The handle of the Cyclone unit that will toggle its output power |
| @returnvalue | True if the operation completed successfully<br><br>False otherwise |

# 5 List of Error Codes

The following error codes are returned from the “get_last_error_code” function call in the SDK. Note the “0x” prefix indicates that the corresponding error code is in hexadecimal format.

**Success**:
0x0000: Programming operations completed without error.

**Application handling related error codes**:
0x00A0: BM is not pre-configured in the Cyclone PRO.
0x00A1: BR is not pre-configured in the Cyclone PRO.
0x00A2: EB is not pre-configured in the Cyclone PRO.
0x00A3: EW is not pre-configured in the Cyclone PRO.
0x00A4: EM is not pre-configured in the Cyclone PRO.
0x00A5: PB is not pre-configured in the Cyclone PRO.
0x00A6: PW is not pre-configured in the Cyclone PRO.
0x00A7: PM is not pre-configured in the Cyclone PRO.
0x00A8: VM is not pre-configured in the Cyclone PRO.
0x00A9: VR is not pre-configured in the Cyclone PRO.
0x00AA: VC is not pre-configured in the Cyclone PRO.
0x00AB: USER1 is not pre-configured in the Cyclone PRO.
0x00AC: USER2 is not pre-configured in the Cyclone PRO.
0x00AD: USER3 is not pre-configured in the Cyclone PRO.
0x00AE: USER4 is not pre-configured in the Cyclone PRO.
0x00AF: USER5 is not pre-configured in the Cyclone PRO.
0x00B0: USER6 is not pre-configured in the Cyclone PRO.
0x00B1: Wrong USER function specified.
0x00B2: PT is not pre-configured in the Cyclone PRO.

0x00B4: Error during power off target.
0x00B5: Error during power on target.

0x00BE: Wrong command line parameters specified.
0x00BF: Specified COM port is not available.
0x00C0: Specified USB port is not available. Please make sure the USB port is available, and the USB cable is connected.
0x00C1: Specified Ethernet IP address is incorrect.
0x00C7: The Cyclone PRO device is not ready. Please check power and connections.

**Execution related error codes**:
0x0001: Unable to detect target communication speed.
0x0002: Unable to enter debug mode.

0x0003: Operation cancelled by user.
0x0004: Error writing data byte block.
0x0005: Error writing data byte block.
0x0006: Error during execution.
0x0007: Error enabling module.
0x0008: Error enabling module.
0x0009: Error enabling module.
0x000A: Error testing target timing.
0x000B: Error reading data byte block.
0x000C: Error un-securing 9S12 target.
0x000D: Error un-securing 9S12 target.
0x000F: Stand alone operations for the Cyclone PRO are not configured.

0x1001: Blank_Check_Word Algorithm is not supported.
0x1002: Blank_Check_Byte Algorithm is not available.
0x1003: Error during blank checking device using blank_check_byte algorithm.
0x1004: Error during blank checking device using blank_check_word algorithm.

0x2003: Error during erasing device.
0x2004: Error during erasing device.

0x3001: Program_Word Algorithm is not supported.
0x3002: Program_Byte Algorithm is not available.
0x3003: Error during programming device.
0x3004: Error during programming device.
0x3005: Error during programming device.
0x3006: Error during programming device.

0x5003: Error during verifying module.
0x5004: Error during verifying range.

0x6003: Error during user functions.
0x6004: Error during user functions.

0x7003: Error calculating trim value.
0x7004: Trim value is not calculated.
0x7005: Program Word Algorithm is not supported for writing the trim value.
0x7006: Program Byte Algorithm is not available for writing the trim value.
0x7007: Error during programming the trim value.
0x7008: Error during verify trim value
0x7009: Trim value is $00 or $FF

# 6 Cyclone Launch

The Cyclone Automated Control SDK includes an application that controls Cyclone operations through the use of simple script files. This application is very easy to set up and use and offers functionality very similar to the DLL.

To find the Cyclone Launch application, navigate to the following directory:

[INSTALLDIR]\Cyclone Launch\

The Cyclone Launch software performs all operations specified in a simple ASCII script file written by the user. A separate batch file would typically be used to launch the utility with the correct parameters.

## 6.1 Startup

a) Connect all Cyclone units to the PC via RS232, USB, or Ethernet. Any combination of different connections is allowed. The exception is that only one RS232 serial port connection can be used; no more than one Cyclone can be connected via the RS232 serial port.

b) Connect all Cyclones to their target systems. This is done using a ribbon cable that connects from the Cyclone to a debug header on the target board.

c) Power up the PC, all Cyclone units, and all target systems that require external power.

d) Run the software from the DOS prompt. Allowed command line parameters are:

```
CYCLONE_LAUNCH.exe [script filename] [?] [!] [-O] [output
filename]
```

Where `[script filename]` is a configuration file containing all operations to be carried out. Refer to section 6.2 for more details. This must be the first parameter passed to the application.

`[?]` designates that Cyclone Launch should remain open when operations are completed. If this parameter is not specified, the utility will automatically close upon completion.

`[!]` designates that Cyclone Launch should remain open if any error is encountered. If there is no error, the software will automatically close upon completion.

`[-O]` designates that the results are saved in an output file following this parameter. The [output filename] parameter is mandatory if [-O] is used.

`[output filename]` is an output file that contains the operation results. A user can check the output file to see if the operations were successful and also identify which Cyclone units have failed.

## 6.2 Command-Line Parameter Examples

```
CYCLONE_LAUNCH.exe  config_script.cfg  !
```

The Cyclone executes all operations specified in the “config_script.cfg” file. The CYCLONE_LAUNCH application remains open if any error occurs.

```
CYCLONE_LAUNCH.exe  config_script.cfg  -O  output_log.txt
```

The Cyclone executes all operations specified in the “config_script.cfg” file and logs all results to the “output_log.txt” file.

## 6.3 Configuration Script File

The configuration file contains two types of commands. The first type is SETUP commands which configures the communication port between the PC and the Cyclone units. The second type is Operation commands which carry out the pre-configured Cyclone operations.

Comments are allowed in the script files. All lines beginning with the semi-colon character ; are treated as comments by the application.

### 6.3.1 SETUP Commands

**SETLOCALIP=ipaddress**

Only needed if the host PC has more than one network card. “ipaddress” indicates the IP address of the network card that should be used during communications. If this command is used, it should be the first command in the script file. “ipaddress” should be in the format of xxx.xxx.xxx.xxx, where xxx=0..255.

**SETTIMEOUT=timeout**

Specifies the maximum amount of time that a Cyclone unit can execute programming operations before a timeout error will occur. “timeout” is specified in number of seconds, and must be greater than zero. If this command is not used, then no timeout errors will occur. This command is useful for safely recovering if a fatal error occurs (e.g. loss of communications, accidental reset or power off on the Cyclone, etc.) Timeout errors can occur during the “ALLSTART” and “START” operation commands (see sections 6.2.2 and 6.2.3)

**OPENTYPE=x**

Specifies whether Cyclone units will be identified by IP address or by device name. Both of these Cyclone parameters are reconfigurable by the user.

The value of x may be:
**IP** - Cyclone units will be identified by their IP addresses
**NAME** - Cyclone units will be identified by their device names

**PORT=y**

Specifies the port that should be used for contacting all subsequent Cyclone units.

The value of y may be:

**USB** - Subsequent Cyclones will be contacted via USB
**ETHERNET** - Subsequent Cyclones will be contacted via Ethernet
**SERIAL** - Subsequent Cyclones will be contacted via Serial

**CYCLONE=identifier**

Connects to a Cyclone unit with the specified identifier. If OPENTYPE=IP, the identifier should be in the format of xxx.xxx.xxx.xxx, where xxx=0..255. If OPENTYPE=NAME, the identifier should be the name of the Cyclone unit.

This command must be used once for each Cyclone that will be controlled. The newly connected Cyclone will also become the selected Cyclone. (See the “SELECT [identifier]” command) This is useful for setting the image number of the Cyclone immediately after connecting to it.

### 6.3.2 Operation Commands that Control All Connected Cyclones

These commands affect all Cyclone units connected by the "CYCLONE=identifier" setup command. Many of these commands are ideal to use when multiple Cyclone units have identical images and configuration.

**ALLIMAGENUM=n**

Sets the image number to be executed on all connected Cyclone units.

**ALLSTART**

Executes stand-alone programming operations on all connected Cyclones.

**ALLERASEIMAGES**

Same as "ERASEIMAGES" except that all connected Cyclone units are erased.

**ALLADDIMAGE [sap_file]**

Same as "ADDIMAGE [sap_file]" except that the image is added to all connected Cyclone units.

### 6.3.3 Operation Commands that Control a Single Cyclone

These commands affect a single connected Cyclone unit. The “SELECT [identifier]” command must first be used to select the appropriate Cyclone prior to using the other commands in this section.

**SELECT [identifier]**

Selects the Cyclone unit that will be controlled until the next “SELECT [identifier]” command.

**IMAGENUM=n**

Sets the image number to be executed on the selected Cyclone unit. This command should be used when a Cyclone unit stores 2 or more images. By default, a Cyclone unit will execute image #1. The valid range of n is between 1 and the number of images stored on the Cyclone unit.

**START**

Executes stand-alone programming operations only on the selected Cyclone unit.

**ERASEIMAGES**

Erases all stand-alone programming images stored on the selected Cyclone unit.

**ADDIMAGE [sap_file]**

Adds a specified image to the selected Cyclone unit. [sap_file] is the full path to the stand-alone programming image file. These files have a .SAP extension and are created with the Cyclone Image Creation Utility.

**PUTDYNAMICDATA [address] [data]**

Performs programming of dynamic data with the selected Cyclone unit. [address] is the starting memory address. [data] are the bytes of data to be programmed. All values should be in hexadecimal format. No more than 255 bytes may be programmed this way.

A Cyclone unit may only use this command after it has performed its “START” command. (The “ALLSTART” command will also satisfy this requirement)

**READDYNAMICDATA [address] [numbytes]**

Reads dynamic data with the selected Cyclone unit. [address] is the starting memory address. [numbytes] is the number of bytes of data

to read. All values should be in hexadecimal format. No more than 255 bytes may be read in this way.

A Cyclone unit may only use this command after it has performed its “START” command. (The “ALLSTART” command will also satisfy this requirement)

## 6.4 Examples

The configuration (.cfg) file should be a pure ASCII file with one command per line without any comments.

### 6.4.1 Typical Usage

```
; Setup commands
OPENTYPE=IP
SETTIMEOUT=60          ;Configure for 1 minute timeout
PORT=USB
CYCLONE=192.168.1.1

; Operation commands
ALLSTART
```

This example connects to a single Cyclone via the USB port and executes its first image (by default, since we never used the “IMAGENUM=n” command). This is the most common usage of the Cyclone Control Utility.

### 6.4.2 Controlling Multiple Cyclones

```
; Setup commands
OPENTYPE=IP
SETTIMEOUT=60          ;Configure for 1 minute timeout

PORT=USB
CYCLONE=192.168.1.1
IMAGENUM=2

CYCLONE=192.168.1.2
IMAGENUM=2

PORT=ETHERNET
CYCLONE=192.168.1.3
IMAGENUM=3

; Operation commands
ALLSTART
```

This example connects to three separate Cyclone units. Two units are connected via USB (192.168.1.1 and 192.168.1.2) and the third is connected via Ethernet (192.168.1.3). The two Cyclone units connected via USB are configured to execute image #2, while the third Cyclone is configured to execute image #3.

### 6.4.3 Programming dynamic data

```
; Setup commands
OPENTYPE=NAME
SETTIMEOUT=60         ;Configure for 1 minute timeout
PORT=SERIAL
CYCLONE=PE_PRO1
IMAGENUM=1

; Operation commands
START
PUTDYNAMICDATA 1080 45 44 49 53 4F 4E
```

Here, a Cyclone is connected via the Serial port and is identified by name rather than IP address. After executing image #1, we write 6 bytes of dynamic data starting at address 0x1080. Note that all parameters for the PUTDYNAMICDATA command should be hexadecimal values.

### 6.4.4 Executing more than 1 image on the same Cyclone

```
; Setup commands
OPENTYPE=NAME
SETTIMEOUT=60        ;Configure for 1 minute timeout

PORT=ETHERNET
CYCLONE=PE_PRO1
IMAGENUM=1

CYCLONE=PE_PRO2
IMAGENUM=1

; Operation commands
ALLSTART

SELECT PE_PRO1
IMAGENUM=2
START
```

Two Cyclone units are connected via Ethernet and both Cyclone units execute their first image. Afterwards, the Cyclone with name “PE_PRO1” will execute its second image.

This example is useful when the processor’s code is split into two separate images. For example, one image could contain the bootloader while the second image contains the main application code.

### 6.4.5 Image Management

```
; Setup commands
OPENTYPE=IP
SETTIMEOUT=60          ;Configure for 1 minute timeout

PORT=USB
CYCLONE=209.61.110.143
CYCLONE=209.61.110.144

; Operation commands
ALLERASEIMAGES
ALLADDIMAGE C:\Images\Image1.SAP
```

In this last example, two Cyclone units connected via USB have their images erased and a new image is added to both Cyclone units. This type of script should only be executed when images need to be updated. A separate script should be used to execute the images stored on the Cyclone.

## 6.5 DOS Error Codes

If the Cyclone Control Utility encounters a fatal configuration error, such as not being able to contact a specified Cyclone or providing invalid commands in the configuration script, the utility will halt immediately.

However, if a Cyclone has an error while it is executing an image, the utility will still continue to process script commands for all other Cyclone units that do not have errors.

DOS error returns are provided so they may be tested in .BAT files. Although this utility controls multiple Cyclone units, the error codes returned only reflect the most recent error. The error codes used are:

0x0000: Success!

Application handling related error codes:
0x00A0: BM is not pre-configured in the Cyclone PRO.
0x00A1: BR is not pre-configured in the Cyclone PRO.
0x00A2: EB is not pre-configured in the Cyclone PRO.
0x00A3: EW is not pre-configured in the Cyclone PRO.
0x00A4: EM is not pre-configured in the Cyclone PRO.
0x00A5: PB is not pre-configured in the Cyclone PRO.
0x00A6: PW is not pre-configured in the Cyclone PRO.
0x00A7: PM is not pre-configured in the Cyclone PRO.
0x00A8: VM is not pre-configured in the Cyclone PRO.
0x00A9: VR is not pre-configured in the Cyclone PRO.
0x00AA: VC is not pre-configured in the Cyclone PRO.
0x00AB: USER1 is not pre-configured in the Cyclone PRO.
0x00AC: USER2 is not pre-configured in the Cyclone PRO.
0x00AD: USER3 is not pre-configured in the Cyclone PRO.
0x00AE: USER4 is not pre-configured in the Cyclone PRO.
0x00AF: USER5 is not pre-configured in the Cyclone PRO.
0x00B0: USER6 is not pre-configured in the Cyclone PRO.
0x00B1: Wrong USER function specified.
0x00B2: PT is not pre-configured in the Cyclone PRO.

0x00B4: Error during power off target.
0x00B5: Error during power on target.

0x00BE: Wrong command line parameters specified.
0x00BF: Specified COM port is not available.
0x00C0: Specified USB port is not available. Please make sure the USB port is available, and the USB cable is connected.

0x00C1: Specified Ethernet IP address is incorrect.
0x00C7: The Cyclone PRO device is not ready. Please check power and connections.
0x00C8: Error in configuration script.
0x00C9: Error erasing Cyclone images
0x00CA: Error adding a Cyclone image

Execution related error codes:
0x0001: Unable to detect target communication speed.
0x0002: Unable to enter debug mode.
0x0003: Operation cancelled by user.
0x0004: Error writing data byte block.
0x0005: Error writing data byte block.
0x0006: Error during execution.
0x0007: Error enabling module.
0x0008: Error enabling module.
0x0009: Error enabling module.
0x000A: Error testing target timing.
0x000B: Error reading data byte block.
0x000C: Error un-securing 9S12 target.
0x000D: Error un-securing 9S12 target.
0x000F: Stand alone operations for the Cyclone PRO are not configured.
0x1001: Blank_Check_Word Algorithm is not supported.
0x1002: Blank_Check_Byte Algorithm is not available.
0x1003: Error during blank checking device using blank_check_byte algorithm.
0x1004: Error during blank checking device using blank_check_word algorithm.

0x2003: Error during erasing device.
0x2004: Error during erasing device.

0x3001: Program_Word Algorithm is not supported.
0x3002: Program_Byte Algorithm is not available.
0x3003: Error during programming device.
0x3004: Error during programming device.
0x3005: Error during programming device.
0x3006: Error during programming device.

0x5003: Error during verifying module.
0x5004: Error during verifying range.

0x6003: Error during user functions.

0x6004: Error during user functions.

0x7003: Error calculating trim value.
0x7004: Trim value is not calculated.
0x7005: Program Word Algorithm is not supported for writing the trim value.
0x7006: Program Byte Algorithm is not available for writing the trim value.
0x7007: Error during programming the trim value.
0x7008: Error during verify trim value
0x7009: Trim value is $00 or $FF

## 6.6 Sample Batch File

Here is an example of calling the Cyclone Control Utility and testing the error code return in a simple batch file. Sample batch files are given for both Windows 95/98 and also Windows NT/2000/XP.

Windows NT/2000/XP:

```
CYCLONE_LAUNCH test.CFG
if errorlevel 1 goto bad
goto good
:bad
ECHO BAD BAD BAD BAD BAD BAD BAD BAD
:good
ECHO done
```

Windows 95/98:

```
START /W CYCLONE_LAUNCH TEST.CFG
if errorlevel 1 goto bad
goto good
:bad
ECHO BAD BAD BAD BAD BAD BAD BAD BAD
:good
ECHO done
```

# 7 Cyclone Communication Protocols

Included with the Cyclone Automated Control SDK Enterprise Edition are documents describing the serial port RS232 and Ethernet communication protocols used by the Cyclone. Knowing these protocols allows the developer to manually send individual byte packets which will be interpreted by the Cyclone as commands.

This is particularly useful for setups that wish to use multiple serial ports on a single host PC, as the DLL does not support more than one serial port connection.

For more detailed information, please refer to the following directories included with your SDK installation:

[INSTALLDIR]\RS232 Protocol\
[INSTALLDIR]\Ethernet Protocol\